On Concepts for Autonomic Communication Elements

Edzard Hoefig, Bjoern Wuest, Borbala Katalin Benko,
Antonietta Mannella, Marco Mamei, Elisabetta Di Nitto

Abstract— **Autonomic communication aims to reduce complexity and management costs of modern communication services and networks. Autonomic Communication Elements (ACEs) are seen as the basis of autonomic communication systems. A main research task of the CASCADAS project is the definition of a component model for ACEs and the release of an open-source toolkit to simplify creation of autonomic communication systems. This paper presents the current progress of the CASCADAS project in defining the ACE component model. A platform abstraction concept is presented along with the basic structures that will be researched within the project. A basic model for ACEs is introduced and exemplified in regard to its properties and requirements. The idea of a SEE ACE is introduced, which enables a homogeneous and self-similar hosting environment. The paper concludes with a discussion of the model and a presentation of open topics, giving insight into the current state of discussion.**

***Index Terms*—Autonomic Communication, Component Technology, Self-Organisation, CASCADAS**

I. Introduction

Under the project name CASCADAS (**C**omponent-ware for **A**utonomic, **S**ituation-aware **C**ommunications, **A**nd **D**ynamically **A**daptable **S**ervices), 14 partners from industry and academia are investigating the main research threads of autonomic communication and will deliver their research results as an integrated open source toolkit of abstractions, algorithms, and tools. The CASCADAS project is one of four projects funded in the same FET program [1].

CASCADAS will identify, develop, and build upon a new model of distributed components, called ACEs (Autonomic Communication Elements), which have the ability to self-organize autonomously and cooperatively with each other, provide specific user communication services, and adapt the provisioning in an autonomic and specific context-aware manner to social and network contexts. In other words, services will be composed of software components capable of understanding the general and specific context in which they operate (physical, technological, social, user-specific and request-specific) and spontaneously aggregate and organise their activities according to that context. The general objective is to reduce the costs of development and configuration of complex communication services, to leverage the exploitation of distributed computing and communication resources, to increase service quality, to improve service availability and reliability, and to make services more usable in line with user needs and expectations. For a more detailed description of the project, please refer to the project's web pages [2] or to the article by Manzalini and Zambonelli [3], who write about the CASCADAS vision.

This paper describes the current state of work on ACE concepts that are developed to serve as a technical base for implementing autonomic, situation-aware and dynamically adaptable services.

II. Platform Transparency

The ACE component model is supposed to be open in regard to the underlying platform, i.e. operating system, programming language and execution environment for ACEs. Generally it's supposed that ACEs are developed using an intermediate language that is executed by a runtime environment on top of an operating system and hardware. The execution of intermediate language can be realised by interpretation, just-in-time-compilation, or compilation into machine code as part of ACE deployment. The intermediate language, runtime environment, operating system, and hardware are referred to as the *platform layers*.

The Authors would like to acknowledge the European Commission for funding the Integrated Project CASCADAS "Component-ware for Autonomic,Situation-aware Communications, And Dynamically Adaptable Services" (FET Proactive Initiative, IST-2004-2.3.4 Situated and Autonomic Communications) within the 6th IST Framework Program. The paper represents the work and contribution of individual parties involved in the project. The authors also wish to thank other project partners for excellent and fruitful discussions; the contents of this paper have been preliminary agreed and further work is ongoing

E. Hoefig is with the Fraunhofer Institute for Open Communication Systems, Berlin, Germany (phone: ++49-30-3463-7243; fax: ++49-30-3463-8243; e-mail: edzard.hoefig@fokus.fraunhofer.de).

B. Wuest is with the Department of Electrical Engineering and Computer Science, University of Kassel, Germany (bjoern@comtec.eecs.uni-kassel.de).

B. K. Benko is with the Department of Telecommunications, Budapest University of Technology and Economics, Hungary (bbenko@hit.bme.hu).

A. Mannella is with Telecom Italia, Turin, Italy (antonietta.mannella@telecomitalia.it)

M. Mamei is with the Department di Scienze e Metodi dell'Ingegneria, Università di Modena e Reggio Emilia, Italy (mamei.marco@unimo.it)

E. Di Nitto is with the Department di Elettronica e Informazione, Politecnico di Milano, Milan, Italy (dinitto@elet.polimi.it)

The structures introduced by CASCADAS are service providing ACEs and a knowledge network, responsible for collecting, organising, correlating and composing information for and about ACEs. These structures are interdependent: Knowledge networks use ACEs as the main structuring mechanism and ACEs use information as disseminated by the knowledge networks to adapt to changes in the environment. The separation between platform and the structures introduced by the project is depicted in fig. 1.

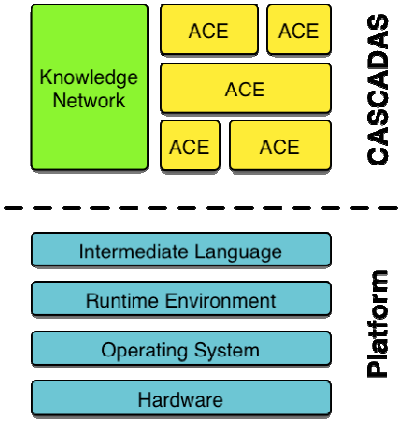

Figure 1: Layer concept showing boundary between platform and the basic structures that are being researched within the project

The CASCADAS project will not do any research in the platform layers. It merely tries to formulate necessary requirements to be fulfilled by a future platform in order to host ACEs. Other projects funded in the same FET program, like ANA or HAGGLE are covering research in this area (see [1] for a description about how these projects complement each other).

A platform delivers the resources needed to manage ACEs, which includes (but is not restricted to) bootstrapping, access, migration, copy, aggregation and retirement of ACEs. Platform resources include at least processing units, data storage, communication facilities and also any other proprietary functionality in hard- or software. We expect the platform to provide late binding and dynamic (un-)loading of code, automated management of its resources (e.g. garbage collection, file handling), introspection mechanisms and access to other operating system services, for example data transport services. These properties are motivated by the need for dynamic behaviour as exposed by ACEs and their ability to provide services in an autonomic and distributed way, for example following the protocol as defined by the authors of [4].

ACEs have access to a so-called *Knowledge Network* that collects, organises, correlates, and composes information**,** enabling the elements to exploit all available information about their situation, however sparse and diverse. Situation is intended here as a generalization of context, relating to both *(i)* the social-organisational context from which services are invoked (i.e., by specific users living in a specific social context and accessing the network with specific devices and network technologies); *(ii)* the technological and physical environment in which ACEs live and execute, primarily their networked environment. This is relevant for the principle of situation awareness, but also represents a common substrate upon which all the other functionality will rely. In fact, it is supposed to be the glue that enables ensembles of ACEs to collaborate in a meaningful and autonomic matter.

III. AUTONOMIC COMMUNICATION ELEMENTS

ACEs are seen as the basic building block for enabling autonomic behaviour for services, providing also the functionality for access and aggregation of services. They are not the service itself, but rather act as an access point to a service, making the service and associated resources available.

A conceptual version of an ACE is depicted in fig. 2. As seen in this diagram, ACEs may be composed in two different ways: externally and internally. Externally composed ACEs collaborate loosely to provide a Service, while every single ACE remains visible as an independent entity. Internal composition aims at providing a service by mutual collaboration of ACEs through binding them more strictly: A new ACE emerges that is representing the whole ensemble and ACEs are found to be "within" this emerging ACE (in this paper the emerging ACE is referenced as *combined* ACE and all aggregated ACEs as *contained* ACEs).

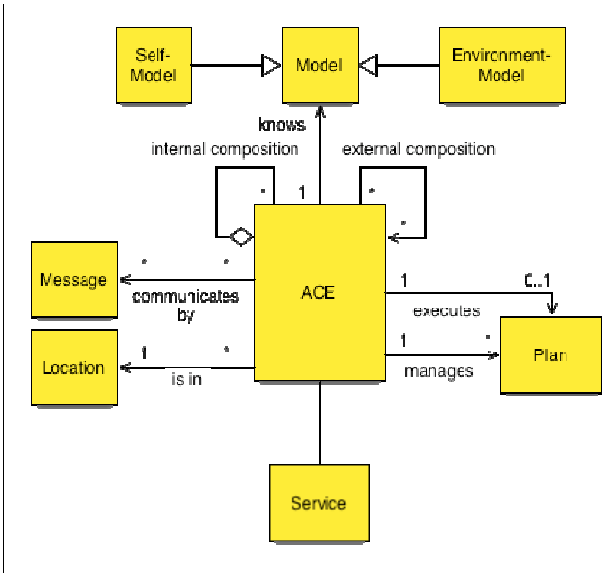

Figure 2: Conceptual UML diagram of the ACE base model. Relationships that are missing cardinalities or navigability are still under discussion

There are at least two models an ACE knows. On the one hand a self-model, representing its own context-dependent behaviour and caring for self-awareness; on the other hand an environment model, used for situation-awareness. The environment model may be the set of available self-models from other ACEs in combination with any other information communicated by the knowledge network. If – in the course

of the CASCADAS project – it turns out that context information is solely generated by other ACEs these two models may collapse into a single one. This seems to be happening in the case of internal composition: the self-model of a combined ACE seems to be the environment-model of all contained ACEs.

Communication between ACEs is message based, facilitating a time and spatial decoupling of the communication partners, for example following mechanisms as surveyed by the authors of [5]. Messages may be buffered when their receivers are unavailable, they could be multiplexed for point-to-multipoint communication, they may be routed for indirect communication between ACEs, and they can be delivered asynchronously avoiding the requirement for synchronisation between ACEs.

Every ACE resides in a location, which marks a position within the knowledge network, giving it access to the stigmergic[1] information within that area. ACEs are supposed to be able to move between locations and are free to migrate to any place where at least one ACE is already in existence.

Furthermore, we identified the notion of a *Plan*, which is an explicit formulation of the way an ACE is supposed to act. ACEs are not only executing plans (which in turn dictates their behaviour), but also manage plans by creating, choosing, changing, rearranging, and removing them and thereby effectively changing their potential behaviour. The planning concept promises to enable adaptation and self-organisation capabilities.

ACEs are structured in two parts. A common part, exposed through a so-called “common interface”, which can be taken to be available with every ACE and a specific part, available through a “specific interface”. This concept is influenced by previous work of some partners as published in [6]. As we don’t expect every ACE to implement the common functionality by itself, a mechanism has to be researched that is able to relay access of an interface transparently to another interface, e.g. by propagating binding information from one ACE to another. This will be of importance when migrating ACEs, using internal composition or creating ACEs based on other ones. A convenient way of creating ACEs would be the cloning of existing ACEs, resulting in inheritance of all common bindings from the parent ACE.

IV. Service Execution Environment (SEE) ACE

SEE ACEs are elements that implement the container functionality used to host other ACEs. They create a homogeneous environment independent of the underlying platform by realising the *common interface* and the functionality described in the *specific interface* of the SEE ACE. Realising the hosting container as an ACE itself ensures the principle of self-similarity and enables the hosting code to benefit from all other functionality available via the common part of ACEs.

SEE functionality is specific to the underlying platform (e.g. certain communication primitives), the code realising it is found in the specific part of an SEE ACE. As every ACE needs to also have a common part, the SEE ACE exports its specific functionality through the common interface. By doing this it enables other ACEs to bind to its specific functionality by using the common interface. This mechanism is depicted in fig 3.

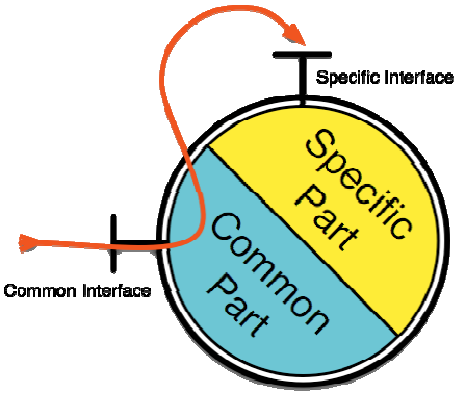

Figure 3: Concept of the Service Execution Environment ACE

As SEE ACEs are highly dependent on the underlying platform it may be assumed that they will not be able to migrate because of their dependence on immobile functionality. Regarding bootstrapping, we will not prescribe a standard way of bootstrapping SEE ACEs, but leave this question open to the decision of implementers. A certain implementation has to decide proprietary on how to supply the initial binding information to the bootstrapped element.

As we expect an SEE ACE to be the first available element on a host, several ACEs would be created with functional bindings depending on this element (e.g. during migration or copy of the SEE ACE). If we consider the case of termination of such an ACE, then a direct consequence would be the invalidation of functional bindings of any dependent ACEs. Finding ways to automatically repair functional bindings, e.g. by redistributing components as proposed in [7], are understood to contribute to the self-repair capability of ACE based systems.

V. ACE component model

Fig. 4 shows the conceptual view of the ACE component model in regard to two ACEs: A SEE ACE and a hypothetical “My” ACE as a placeholder for any other ACE. The *Common Interface* is understood as the access point to the common part, exposing bindings to the functionality that is shared among both ACEs. The *common part* contains bindings to functionality that implements the *Common Interface*. In the case of an SEE ACE it uses the *SEE Interface* to realise the *Common Interface* functionality.

The *SEE Interface* gives an access point to the platform dependent code of the SEE ACE, realised in the *SEE specific*

[1] Stigmergy refers to communication by modification of the environment. It is an often observed strategy in emergent systems.

part. The *SEE Interface* is a *Specific Interface*. *"My" Interface* is also a *Specific Interface* and realised in the *"My" specific part*, which is a *Specific part*. The *"My" ACE* consists of a *Common part* and the *"My" specific part*.

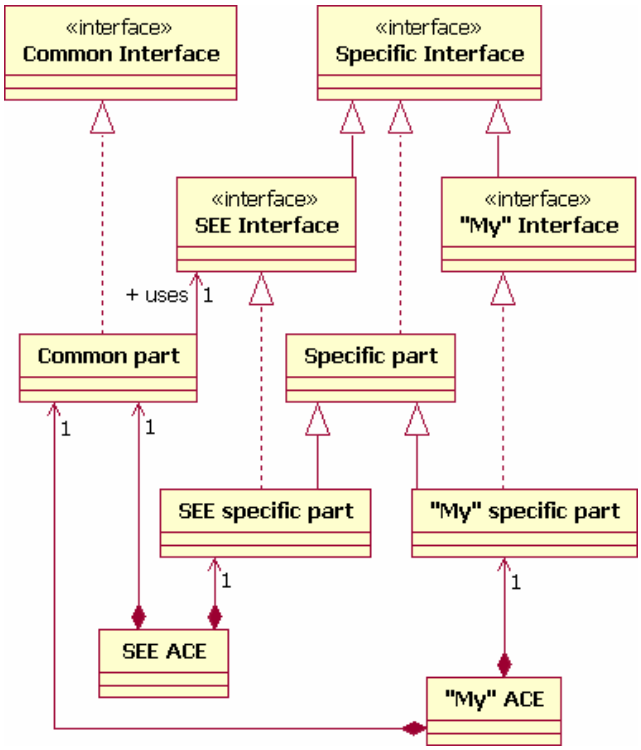

Figure 4: Conceptual view of the ACE component model

Implementing a new ACE, i.e. *"My" ACE*, requires a definition of the *"My" Interface* and its realisation in the *"My" specific part*. If *"My" ACE* would be realised as an internal composition of existing ACEs, then "My" ACE would inherit all bindings from its contained ACEs. It would then be possible for it to substitute existing bindings with appropriate ones from its own specific part. Using this mechanism would give the possibility for dynamic composition of functionality.

VI. Discussion

The ACE component model should enable ACEs to be self-similar, -organising, -managing, -configuring, -aware, -healing and so forth. This section discusses the concepts for the ACE component model in regard to different self-* aspects.

A. Self-Similarity

Self-similarity is about aggregating components while the aggregate is *identical* to its parts. The purpose may be to increase scalability, ease configuration or re-use solutions on different levels of abstraction. An example is a WWW server accessible via a domain name, but actually distributing the load to a number of different computers that service the same content, have the same structure, address, etc.

The aforementioned component model is not only self-similar because it specifies that the common interface of ACEs is not allowed to be modified and has to be implemented by every ACE, but also because composition would allow for an elegant use of group communication, For example consider an aggregated ensemble of 10 *"My" ACE*, whose functionality is accessible via the 10 specific *"My" Interfaces. Using a* composed ACE the same functionality could still be accessible via the *"My" Interface* using group communication primitives, like "one-of" or "all". The composed ACE would then transparently delegate the task to either a single ACE or the whole group. Please refer to [8] on how such primitives may be implemented. Consequently, the ACE Component Model allows the realisation of self-similar services.

B. Self-Organisation

Self-organisation means that the organisation of ACEs is defined by the ACEs themselves. The CASCADAS project has a complete work package devoted to self organisation and we expect that support for the models and strategies developed by this working group would be based on the internal and external composition capability of ACE's, the information accessible by the knowledge network, plus their ability to migrate to other locations.

C. Self-Awareness

Regarding to ACEs, the term *self-awareness* has been used in reference to the concept of introspection and reflection: being able to analyse its own interfaces at runtime. ACEs should be able to analyse any interface, therefore they may also do this with their own interfaces. The project also aims at realising semantically meaningful introspection, leading to the integration of models for describing ACE behaviour and thereby allowing reasoning about it.

D. Self-Healing

Automated repair of broken links between ACEs may be the most prominent candidate for *self-healing* aspects. Looking for new functionality bindings when ACEs are terminated (as described at the end of section IV) or when communication links vanish can be regarded as healing capabilities.

VII. Open Topics

The ACE component model is work in progress. There are a number of topics that are actively discussed within the CASCADAS project.

ACEs provide services. However, there is no clear definition for the exact relationship between ACEs and services. One option is that ACEs are service access points. Another possibility is that services are implemented within ACEs, and as such, ACEs contain services. Another open topic related to this one is the relation between service and resource. If resources are made available by services, then clarifying the relationship between ACEs and services is sufficient. If resources are not made available by services, then the exact relationship between ACEs and resources has to be defined, too.

When ACEs compose internally, the self-model of the combined ACE will be the environment-model of the contained ACEs. The open question is the relation of the models when ACEs compose externally. Externally composed ACEs collaborate loosely and every ACE remains visible as an independent entity. This makes it possible that ACEs collaborate in different externally composed ACEs. It is not clear whether the self-model of an external ACE will be the environment model of the contained ACEs, and what happens if an ACE is contained in two different externally composed ACEs.

ACEs are in a location. This location has to be understood as a concept. Its relationship to real world position (as acquired by e.g. mobile devices using positional equipment like GPS) or structural positions (e.g. an ACE contained within a combined ACE) remains to be discussed. The results of this discussion will likely have an impact on the communication between ACEs, i.e. the sending and addressing of messages between them. In this context we are also discussing about ACE migration and the relationship of functionality to location. Imagine an ACE migrating; it would need to be clarified which functionality will be re-bound, physically migrated with the ACE or relayed to the original location of the ACE.

The SEE ACE has to be the first ACE on any platform. Its purpose is to provide a homogeneous hosting environment for ACEs. Hosted ACEs depend on the SEE ACE due to functional bindings. An open issue here is the handling of ACE lifecycle events, like the shutdown or crash of an SEE ACE. There are two possibilities currently discussed: terminating dependant ACEs (or parts thereof) or moving them to another SEE ACE. The later approach would again justify the usage of the term “self-healing”.

Another topic to be discussed is how ACE’s find, bind and establish communication with each other. Currently it is proposed to use discovery in an ACE-based overlay network. In this case we would also need to consider the establishment of network links, building of a viable network topology, selection of routing algorithms and so forth.

The understanding of “self-awareness” that is used in this paper is rather primitive. We are aware that “self-awareness” means more than just reflection mechanisms, therefore we included a self-model that delivers the resources to cater for self-awareness. Yet, there is no detailed understanding about the properties of the self-model, apart from its relation to the environment model in internal compositions.

VIII. Conclusion

The ACE component model aims at reducing the costs of development and configuration of complex communication services, leveraging the exploitation of distributed computing and communication resources, increasing service quality, improve service availability and reliability, and in making services more usable in line with user needs and expectations.

ACEs are run on top of a platform providing the resources needed to manage ACEs, which includes bootstrapping, access, migration, copying, aggregation, and retirement of ACEs. ACEs will build, and have access to, a *Knowledge Network* that collects, organises, correlates, and composes information. Using this locally available information would enable ACEs to adapt to the requirements of a situation. ACEs can compose to form new ACEs; they are described in models, execute plans, communicate with each other, are in a location, and provide access to a service.

They may compose, either internally or externally and are self-aware by being able to reason about themselves or their environment using models and plans. ACEs are also able to manage and modify their own plans to react to new situations. Communication between ACEs is message based, facilitating a time and spatial decoupling. A special Service Execution Environment (SEE) ACE implements container functionality used to host other ACEs and provides a homogeneous environment for them. Realisation as an ACE itself ensures the principle of self-similarity, which is a major aspect of autonomic communication systems.

A discussion on self-* aspects that are expected to be present in all autonomic communication systems, e.g. self-similarity, self-organisation, self-awareness, and self-healing, shows the viability of the ACE component model. Finally, a number of open topics indicate the next steps to be taken within the CASCADAS project in technically realising the open-source toolkit that will provide abstractions, algorithms, and tools to simplify the implementation of ACEs and to aid in building autonomic communication systems.

References

[1] F. Sestini, “Situated and Autonomic Communication an EC FET European initiative”, ACM SIGCOMM Computer Communication Review, Vol. 36, Issue 2, pp. 17-20, April 2006

[2] CASCADAS project web page, http://www.cascadas-project.org

[3] A. Manzalini, F. Zambonelli, "Towards Autonomic and Situation-Aware Communication Services: the CASCADAS Vision", IEEE Workshop on Distributed Intelligent Systems, Prague (CZ), June 2006 (to be presented)

[4] F. Saffre, H. Blok, ”SelfService: a theoretical protocol for autonomic distribution of services in P2P communities”, Proceedings of the 2nd Workshop on Engineering of Autonomic Systems, pp. 528-534, 2005

[5] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec, “The Many Faces of Publish/Subscribe”, In ACM Computer Surveys, Vol. 35, No. 2, pp. 114-131, June 2003

[6] O. Droegehorn, F. Carrez, K. David, H. Helin, S. Arbanowski *et al.*, "Generic Service Elements and Enabling Middleware Technologies", Wireless World Research Forum (WWRF), Working Group 2, Whitepaper

[7] M. Mikic-Rakic, N. Medvidovic, "Support for Disconnected Operation via Architectural Self-Reconfiguration," Proceedings of the First International Conference on Autonomic Computing (ICAC'04), pp. 114-121, 2004.

[8] C. Reichert, D. Witaszek, "An Implementation of the Group Event Notification Protocol", Fraunhofer FOKUS, Technical Report TR-2002-0301, March 2002